Clasificador Multiclase con Redes Neuronales Convolucionales

Resumen. El área de reconocimiento de imágenes ha cobrado considerable interés en los últimos años. Este trabajo explora arquitectura de redes neuronales para conseguir un modelo capaz de reconocer un conjunto variado de 30 especies de animales en su hábitat natural. Para cumplir el objetivo, se utilizaron técnicas de Transfer Learning y Fine Tuning para adaptar redes neuronales ampliamente usadas que han demostrado ser eficaces y eficientes, tales como VGG-16, ResNet50 e InceptionV3. Previamente, se procesaron las imágenes de un dataset, reduciendo su tamaño y extrayendo la región de interés. Asimismo, se implementaron técnicas para evitar el overfitting como *data augmentation*, *early stopping*, *decay*, *reduceLROnPlateau* y *oversampling*. Los resultados obtenidos fueron satisfactorios, tanto en métricas de error y precisión, como en F1-Score y ROC-AUC, alcanzando valores muy cercanos a 1. Una vez alcanzados estos resultados se realizó un análisis para comprender los puntos débiles de los modelos obtenidos, utilizando *t-SNE* y matrices de confusión. Como conclusión, los principales errores se encuentran entre especies de animales muy similares en formas, tamaños, hábitat, colores y texturas que incluso para un humano sería difícil diferenciar.

Keywords: Deep Learning, Redes Neuronales Convolucionales, Transfer Learning, T-SNE, Hiperparámetros.

1 Introducción

Uno de los problemas más usuales en Inteligencia Artificial y Aprendizaje Profundo [2] es poder entrenar un modelo que sea capaz de aprender características significativas de las distintas imágenes, y luego, poder clasificar una imagen nueva. Para poder desarrollar estos modelos, es necesario que haya un gran volumen de datos (imágenes) para que el modelo pueda extraer y "aprender" qué características posee cada clase que quiera clasificarse. Todo esto es posible gracias a dos aspectos: primero, al gran poder computacional que existe en la actualidad, habiendo incluso herramientas en la nube que brindan procesamiento gratuito en GPU, como la utilizada para implementar este trabajo (Google Colab) con Python [1]; y segundo, a que se pudieron desarrollar Redes Neuronales Convolucionales para una mejor clasificación de las imágenes [3][4]. Estas capas convolucionales permiten extraer una gran cantidad de características de las cuales el modelo podrá tener como dato.

Una vez obtenidas las imágenes [5], por cuestiones de limitaciones de memoria RAM y de tiempos de entrenamiento, se pre-procesan, llevándolas del tamaño original (imágenes 4:3) a un tamaño de 128x128. Con respecto a la etapa de segmentación, medición de características y clasificación en el aprendizaje profundo se ve como una única etapa, en donde se realiza el entrenamiento de la red en su

totalidad. En esta etapa de entrenamiento, las capas convolucionales van extrayendo las características necesarias para poder clasificar cada una de las distintas clases, en nuestro caso, cada una de las 30 especies de animales.

Durante una convolución, los filtros (matrices del mismo tamaño que los mosaicos) se deslizan sobre la cuadrícula del mapa de atributos de entrada de forma horizontal y vertical, un píxel a la vez, como podemos observar en la Figura 1. Durante el entrenamiento, la red neuronal convolucional "aprende" los valores óptimos para las máscaras que le permiten extraer atributos significativos (texturas, bordes, formas) de la imagen. Conforme aumenta la cantidad de filtros que se aplican a la entrada, también aumenta la cantidad de atributos que puede extraer la red. En este trabajo, exploramos y analizamos el uso de técnicas de Transfer Learning y Fine Tuning para adaptar las redes neuronales convolucionales VGG-16 [6], ResNet50 [7] e InceptionV3 [8]. Los resultados obtenidos muestran valores de *accuracy* prometedores que demuestran la viabilidad de nuestro enfoque.

El resto del trabajo se organiza de la siguiente manera. La sección 2 describe la metodología utilizada en el presente trabajo. La sección 3 muestra y analiza los resultados obtenidos. Finalmente, la sección 4 concluye el trabajo e identifica líneas para trabajos futuros.

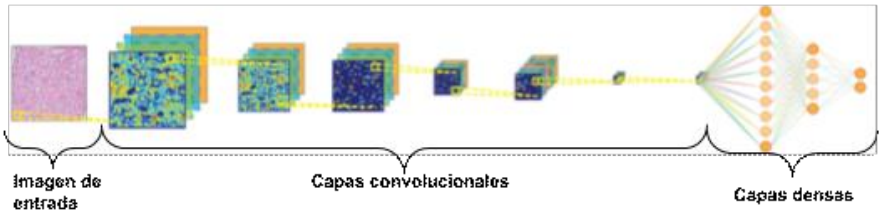

Fig. 1. Pipeline convencional de Redes Neuronales Convolucionales.

2 Metodología

Esta sección describe el dataset utilizado en este trabajo, junto con las arquitecturas de aprendizaje profundo, hiperparámetros y las configuraciones necesarios para optimizar las métricas establecidas.

2.1 Dataset

El dataset utilizado se extrajo de la web Kaggle [5]. Cuenta con 13.000 imágenes que clasifican a 30 tipos de animales diferentes: antílope (1), murciélago (2), castor (3), lince (4), búfalo (5), chihuahua (6), chimpancé (7), collie (8), dálmata (9), pastor alemán (10), oso grizzly (11), hipopótamo (12), caballo (13), orca (14), topo (15), alce

(16), ratón (17), nutria (18), buey (19), gato persa (20), mapache (21), rata (22), rinoceronte (23), foca (24), gato siamés (25), mono araña (26), ardilla (27), morsa (28), comadreja (29) y lobo (30). Además de estas imágenes, la web cuenta con un archivo de imágenes pre-procesadas de tamaño 64x64 y otro archivo conteniendo pesos ya entrenados para este dataset de la red VGG-16

En la Figura 2 se observa la distribución del dataset, y se verifica que es desbalanceado. La cantidad de muestras de cada especie de animal no es homogénea para cada clase, hay clases con más de 900 imágenes y otras con menos de 100. El pre-procesamiento de las imágenes del dataset de entrada resuelve, entre otras limitaciones, el uso de memoria RAM y mejora los resultados. Consiste en transformar cualquier imagen del dataset de entrada para que su tamaño sea de 128x128 pixeles.

Para aumentar la cantidad de muestras del dataset se utilizaron métodos de aumento de datos (Data Augmentation), en donde se generan nuevas imágenes aplicando transformaciones lineales a las existentes, lo que ayudará a obtener mejores predicciones de las nuevas imágenes a clasificar. Se aplicó la misma de dos maneras distintas, Data Augmentation tradicional y Oversampling [9]. La primera consiste en distintas transformaciones lineales a las imágenes de entradas para generar nuevos ejemplares, mientras que la segunda aplica data augmentation para nivelar hacia arriba el número de imágenes cuya clases tienen la menor frecuencia.

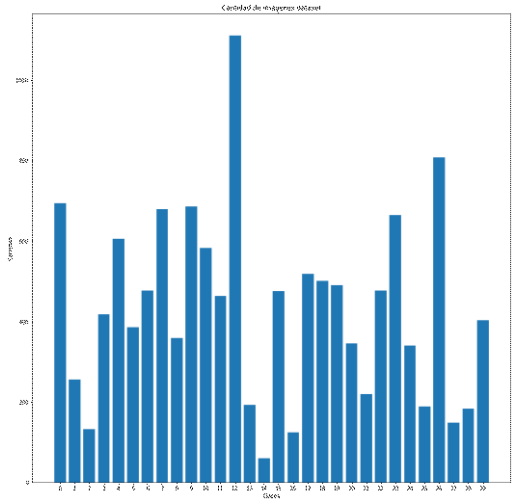

Fig. 2. Distribución de imágenes por clase de animal.

2.2 Hiperparámetros

Esta sub-sección describe los diferentes hiperparámetros y configuraciones utilizados en este trabajo.

2.2.1 Función de activación

Tiene como objetivo tomar los pesos calculados por la capa anterior, modificarlos según corresponda (relu, sigmoid, tanh, etc.) y pasarlos a la siguiente capa como valores de entrada. Se utilizaron tres funciones de activación distintas: Relu donde si la entrada es negativa o cero, el resultado es 0 y además si la entrada es positiva, el resultado es igual a la entrada [10]; Sigmoid [10], la cual asigna resultados de regresión multinomial o logística a probabilidades, y devuelve un valor entre 0 y 1; y Softmax [10] que siempre estará en la última capa de la red ya que transforma los valores recibidos en un vector de probabilidades del tamaño de la cantidad de clases, siendo en este caso un vector de 30 probabilidades que significan las probabilidades que la imagen recibida pertenezca a cada clase.

2.2.2 Tratamiento del Learning Rate

Quizás el hiperparámetro más importante de una red neuronal sea el Learning Rate [11][12], que establece cuán agresivo será el aprendizaje. Éste determina el tamaño del paso en cada iteración mientras se mueve hacia un mínimo de una función de pérdida. Dado que influye en qué medida la información recién adquirida anula la información antigua, representa metafóricamente la velocidad a la que un modelo de aprendizaje automático "aprende" y por ello la modificación de este valor es muy importante en el aprendizaje profundo. Dado su relevancia en los resultados, se ha estudiado la modificación de este hiperparámetro dinámicamente durante el entrenamiento del modelo utilizando distintas opciones, como el Decay, ReduceLROnPlateau [13] y Regularización L1 y L2 [14].

2.3 Manejo del sobre-entrenamiento

Para evitar el sobre-entrenamiento [15], se utilizaron técnicas basadas en ir disminuyendo el Learning Rate (Tasa de aprendizaje) para poder llegar a mínimos globales más rápidamente como Decay, ReduceLROnPlateau y Regularización. Además, se utilizaron capas densas de *Dropout*, la cual consiste en desconectar conexiones entre capas perdiendo información, lo cual hace que aprenda menos de las clases predominantes.

2.4 Exploración de arquitecturas de redes convolucionales

Para la realización del modelo se utilizó una técnica llamada *transfer learning* [16][17]. De esta manera solo es necesario enfocarse en las capas finales de la red y en los hiperparámetros, realizando *fine tuning* [18] sobre el modelo. Para ello, utilizamos Keras [19], una biblioteca que brinda muchas facilidades a la hora de implementar esto en Python, con redes ya creadas y entrenadas en base al conocido dataset *ImageNet* [20]. Este dataset contiene más de 14 millones de imágenes etiquetadas sobre 1000 clases, varias de ellas animales, lo que hace que sean perfectas para este ejemplo. Para la implementación, se debió crear un nuevo modelo de Keras, agregar la red convolucional (VGG16, ResNet50 o InceptionV3) y luego agregar las capas densas con los hiperparámetros establecidos, tal como se muestra en la Figura 3 y la Figura 4.

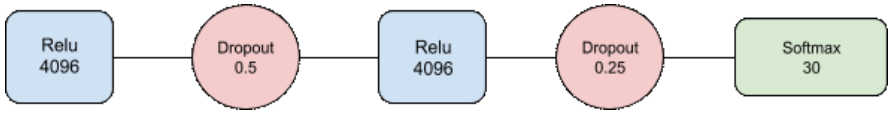

Fig. 3. Esquema de capas densas de modelo VGG versión 1.

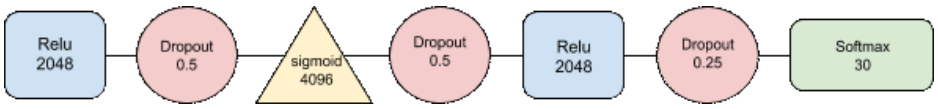

Fig. 4. Esquema de capas densas del modelo VGG16 versión 2, ResNet50 e InceptionV3.

Las métricas usadas para medir el rendimiento y eficacia de los modelos fueron las siguientes: Categorical_crossentropy loss [21], Accuracy, TopKCategoricalAccuracy [22], AUC [23], y F1-Score [24].

2.5 Visualización de resultados: t-SNE

Poder realizar una visualización de la distribución de los datos es sumamente útil para comprender cómo hace el algoritmo para reconocer cada una de las clases. En consecuencia, utilizamos t-SNE [25], una técnica no lineal no supervisada utilizada principalmente para la exploración de datos y la visualización de datos de alta dimensión.

3 Resultados Experimentales

En esta sección se describen y analizan los resultados obtenidos al clasificar las imágenes del dataset. Para ellos, se realizaron 3 experimentos, uno para cada arquitectura convolucional: VGG16, ResNet50 e InceptionV3. Para armar el dataset de training se extrajo el 85% de las imágenes de cada clase de animal, y el 15% restante se usó para armar el dataset de testing. En el caso de VGG16 mostraremos 2 versiones diferentes, en las cuales se puede observar notoriamente la mejora de la red a través del *fine tuning* realizado. Se tomó como notación para las métricas que al referirse a los resultados durante el proceso de entrenamiento, se utilizará su abreviación, mientras que para referirse al conjunto de testing utilizaremos el prefijo "val_" continuado con su abreviación.

3.1 Experimento 1: VGG16

Los primeros experimentos con VGG16 utilizan los pesos sugeridos por Kaggle y el arreglo NumPy [26] con las imágenes provistas por el dataset, de 64x64 pixeles. No se aplicó ningún método de aumento de datos (Data Augmentation). Tampoco se utilizó el ReduceLROnPlateau con el que el *learning rate* se modifica si la *val_loss* no mejora después de cierta cantidad de épocas. Además este modelo posee la arquitectura de capas densas como se muestra en la Figura 3. La red se entrenó con 40 épocas pero podemos ver que empezó a sobre-entrenarse alrededor de la época 20,

como se puede ver en la Figura 5 (a). También podemos ver el gráfico de la métrica AUC en la Figura 5 (b) dejando evidencia que se encuentra muy separado con respecto a la función del entrenamiento (AUC) y que su valor ronda alrededor de 0.9.

Luego testeamos el funcionamiento de la red con el 15% de imágenes que habíamos separado del dataset para testing. Y los resultados fueron los siguientes:

- Test accuracy: 47,74%
- Top 3 accuracy: 68,76%
- F1-Score: 0.4079%

En busca de mejores resultados se aplicó la técnica de transfer learning, inicializando los pesos de la VGG16 con los pesos de "ImageNet" y entrenando las últimas capas de la misma para que dichos pesos se ajuste al problema de clasificación de animales. Además se probó haciendo Oversampling y luego Data Augmentation sin nivelar las clases y sorprendentemente los resultados obtenidos fueron mejores en este último caso.

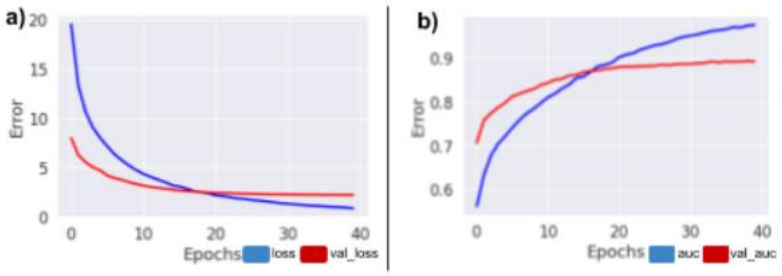

Fig. 5. a) Avance de la pérdida a lo largo del entrenamiento; b) Avance del valor AUC a lo largo del entrenamiento.

Seguimos pensando en el pre-procesamiento de las imágenes y nos dimos cuenta que en las versiones anteriores se estaba forzando a la imagen a ser de 128x128, lo cual genera deformaciones grandes en ciertas imágenes y que en algunos casos nos era imposible darnos cuenta de qué animal se trataba, incluso para nosotros. Para arreglar dicho problema, hicimos un *resize* del menor de los ejes a 128 manteniendo el *ratio* y luego hicimos un *crop* centrado. De esta forma la imagen no pierde calidad y de igual manera la achicamos a un tamaño de 128x128. Otra técnica que incluimos para evitar el overfitting fue la regularización explicada anteriormente. La arquitectura de las capas densas se puede observar en la Fig. 4. Entrenamos la red 100 épocas utilizando ReduceLROnPlateau por si la val_loss convergía, y como se puede ver en la Figura 6 (a) y 6 (b) tanto la curva de la pérdida como la curva AUC, arrojaron mejores resultados. Luego testeamos el funcionamiento de la red con el 15% de imágenes que habíamos separado del dataset para testing. Y los resultados fueron los siguientes, superando ampliamente los obtenidos en la primera versión:

- Test accuracy: 84,41%
- Top 3 accuracy: 93,79%
- F1-Score: 0.7862%

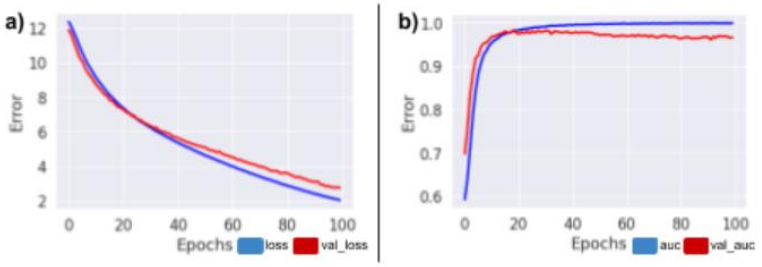

Fig. 6. (a) avance de la loss a lo largo del entrenamiento; (b) avance del valor AUC a lo largo del entrenamiento.

3.2 Experimento 2: ResNet50

Al momento de comenzar a probar la ResNet50, ya habíamos obtenido muy buenos resultados con la VGG16, y no se cometieron los mismos errores. Por ende, las pruebas que se hicieron fueron detalles a los parámetros del modelo, ajustando el *learning rate*, el *batch_size*, la cantidad de épocas, la regularización y las capas densas finales. Por lo que pudimos experimentar, un *batch_size* alto funcionó mejor en la VGG16 mientras que un *batch* menor generó mejores resultados en la ResNet; utilizar regularización en la ResNet no fue igualmente de efectivo que en la VGG16, haciendo que tarde mucho más tiempo el entrenamiento sin reflejar buenos resultados en la red. Al no utilizar regularización, se pudo disminuir la cantidad de épocas a entrenar; el learning rate utilizado en la ResNet fue un factor 0.1 mayor. En cuanto a la arquitectura de las capas densas se respeta la misma que en la Figura 4.

El comportamiento de la curva que se muestra en la Figura 7, muestra que en ningún momento hay overfitting gracias a las herramientas utilizadas para que esto no suceda. Sin embargo, viendo esto se puede decir que la red tiene más potencial y realizando ciertos cambios se podría llegar a mejores resultados alargando el descenso de la pérdida (loss).

Con respecto al accuracy y el F1-score del test, los resultados son esperados teniendo en cuenta la comparación con la red VGG16, ya que comparándola con los resultados obtenidos sobre el dataset ImageNet, la anteriormente mencionada obtuvo un porcentaje menor de acierto que la ResNet:

- Test accuracy: 87,12%
- Top 3 accuracy: 96,46%
- F1-Score: 0.8201%

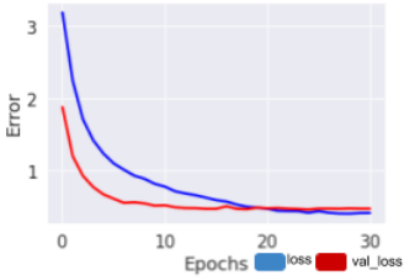

Fig. 7. Avance de la pérdida (loss) a lo largo del entrenamiento.

3.3 Experimento 3: InceptionV3

Se realizaron pruebas de todo tipo en la red Inception. Se cambió varias veces el learning rate, se probó con *batches* grandes, chicos y medianos, se agregaron capas densas a la red, se disminuyó el dropout, entre muchas otras configuraciones, pero a pesar de ello, no se superaron los resultados de la ResNet50. Nuevamente para este modelo convolucional, utilizamos las capas densas descritas en la Figura 4. Debido a los puntajes en el dataset ImageNet, se creía que la InceptionV3 obtendría mejores resultados que las otras dos arquitecturas, pero al momento en que llegaba a un accuracy de 84 u 85%, el entrenamiento se estancaba demasiado. Solo se pudo mejorar un 1% los resultados, alcanzando un accuracy de 86% con la siguiente configuración: pesos entrenables, inicializados con los utilizados para la ImageNet; las mismas capas densas mencionadas en las otras arquitecturas; LR=0.0001, Decay=1e-7; Loss=categorical_crossentropy; ReduceLROnPlateau sobre val_loss con factor= 0.5 y , en este experimento se usó épocas=2; Regularización L1 (lineal) = 0.0001; Batch_size=32; Épocas= 50. Los resultados obtenidos por este modelo testeando en las 1950 imágenes de testing que tenemos separadas fueron los siguientes:

- Test accuracy: 86,41%
- Top 3 accuracy: 95,07%
- F1-Score: 0.8097%

3.4 Resultados obtenidos con t-SNE

A continuación, mostraremos los resultados obtenidos por cada modelo implementado. Cabe aclarar que debido a la forma de realizar la reducción de dimensionalidad t-SNE, cada imagen es un espacio distinto y se deben mirar por separado, no como un progreso entre versiones. Analizando las diferentes visualizaciones t-SNE de los modelos ilustradas en la Figura 8, se puede distinguir cómo mejoró la clasificación de las imágenes del modelo VGG16 versión 1 hasta la ResNet50 que fue la red que mejor resultados arrojó (accuracy: 87%).

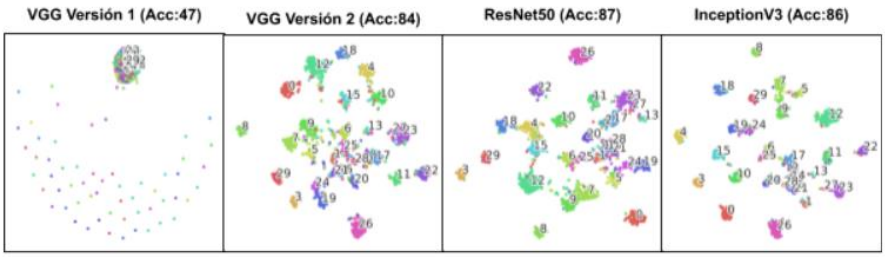

Fig. 8. Gráfico T-SNE de cada uno de los modelos implementados.

4 Conclusiones

Luego de haber analizado los resultados podemos afirmar que hubo un gran avance en los modelos desarrollados desde la versión inicial utilizando VGG16 hasta las versiones finales de las tres arquitecturas. Los tres modelos finales obtienen muy buenos resultados, tanto en su accuracy (Top-1 y Top-3) como en la métrica F1-score, siendo la ResNet50 la ganadora por un pequeño porcentaje. Esto no fue lo esperado, siguiendo los puntajes que se obtuvieron con el dataset original de ImageNet, pero era esperable ya que éste dataset cuenta con 333% más de clases. Durante el transcurso del trabajo nos encontramos con gran cantidad de dificultades, como la baja dimensión de las imágenes pre-procesadas que incluía el dataset, el desbalance del mismo, los malos resultados del transfer learning clásico, entre otros.

Como trabajo futuro, creemos que estos resultados podrían mejorarse si se tendría un dataset más amplio y con cantidad de imágenes balanceadas. Además creemos que debido a las limitaciones que tuvimos con la RAM no pudimos pre-procesar las imágenes para que tengan un tamaño de 224x224, y que si entrenamos con imágenes de ese tamaño los resultados serían mejores a los actuales, principalmente debido a que las arquitecturas convolucionales elegidas tienen este tamaño de entrada.

Referencias

1. Python, https://www.python.org/
2. J. Schmidhuber, "Deep Learning in Neural Networks: An Overview" http://arxiv.org/abs/1404.7828, 2014
3. Numerentur.org, "CNN-RNA Convolucional"
4. Ciresan, Dan; Ueli Meier; Jonathan Masci; Luca M. Gambardella; Jurgen Schmidhuber (2011). "Flexible, High Performance Convolutional Neural Networks for Image Classification"
5. Dataset Kaggle, https://www.kaggle.com/vic006/beginner#sample_submission.csv
6. VGG-16, https://arxiv.org/pdf/1409.1556.pdf
7. ResNet50, https://arxiv.org/abs/1512.03385

8. InceptionV3, https://arxiv.org/abs/1512.00567
9. Data augmentation y oversampling, https://arxiv.org/pdf/1609.08764.pdf
10. Funcion de activacion, https://keras.io/api/layers/activations/
11. Murphy, Kevin P. (2012). Machine Learning: A Probabilistic Perspective. Cambridge: MIT Press. p. 247.
12. Hafidz Zulkifli (21 January 2018). "Understanding Learning Rates and How It Improves Performance in Deep Learning". Towards Data Science. Retrieved 15 February 2019.
13. ReduceLROnPlateau, https://keras.io/api/callbacks/reduce_lr_on_plateau/
14. Regularizadores en keras, https://keras.io/api/layers/regularizers/
15. Tetko, I. V.; Livingstone, D. J.; Luik, A. I. (1995). "Neural network studies. 1. Comparison of Overfitting and Overtraining"
16. Lin, Yuan-Pin; Jung, Tzyy-Ping (27 June 2017). "Improving EEG-Based Emotion Classification Using Conditional Transfer Learning". Frontiers in Human Neuroscience. 11: 334.
17. West, Jeremy; Ventura, Dan; Warnick, Sean (2007). "Spring Research Presentation: A Theoretical Foundation for Inductive Transfer"
18. Käding, Christoph & Rodner, Erik & Freytag, Alexander & Denzler, Joachim. (2017). Fine-Tuning Deep Neural Networks in Continuous Learning Scenarios. 588-605. 10.1007/978-3-319-54526-4_43.
19. Keras, https://keras.io/
20. ImageNet, http://www.image-net.org/
21. Categorical crossentropy, https://keras.io/api/losses/probabilistic_losses/#categoricalcrossentropy-class
22. TopKCategoricalAccuracy, https://keras.io/api/metrics/accuracy_metrics/
23. AUC, https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc?hl=es-419
24. F1-Score, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
25. Visualizing Data using t-SNE Laurens van der Maaten, Geoffrey Hinton; 9(Nov):2579--2605, 2008 http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf
26. NumPy, https://numpy.org/